FOOD & BIOBASED RESEARCH
WAGENINGEN UR

State of the art in federated Querying in SPARQL

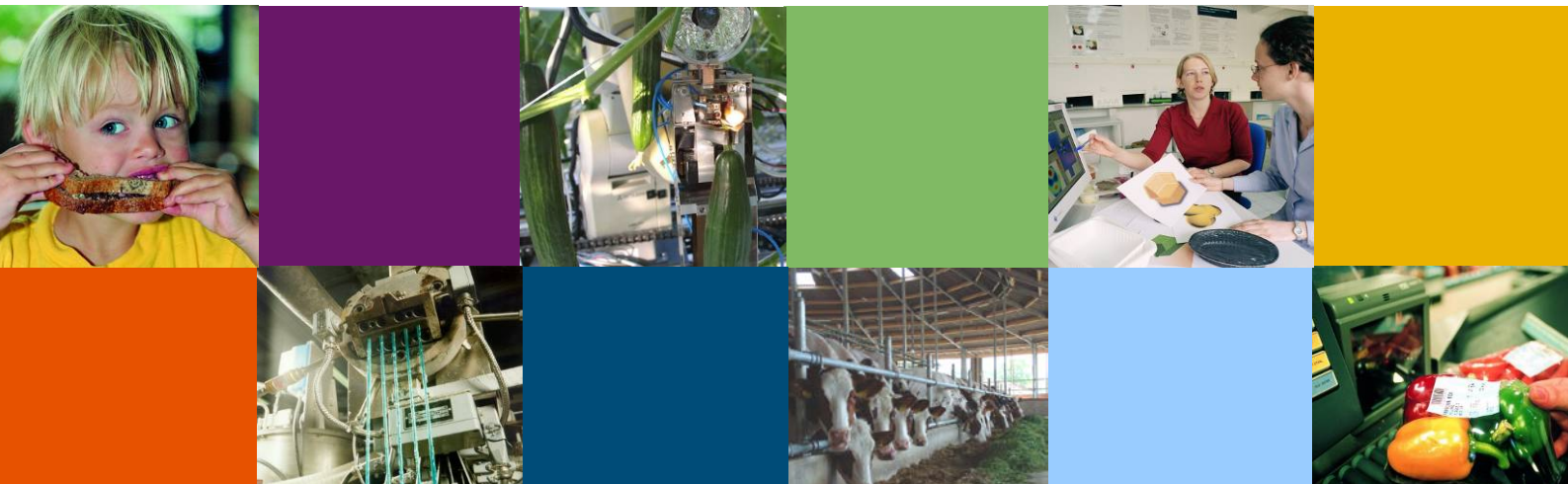
Deliverable Q3 2014

eFoodlab WP1

Mari Wigham

COMMIT/

Rapport nr. 1522



Colophon

Title	State of the art in federated querying in SPARQL
Author(s)	M. Wigham
Date of publication	13-10-2014
Confidentiality	No
Approved by	Jan Top

Wageningen UR Food & Biobased Research
P.O. Box 17
NL-6700 AA Wageningen
Tel: +31 (0)317 480 084
E-mail: info.fbr@wur.nl
Internet: www.wur.nl

Contents

1 Background

In COMMIT/ eFoodLab we are carrying out research into how we can better find and share knowledge about food. There are diverse data sources in existence in the food community, made available by various different parties, from government regulators to commercial businesses, via many different sorts of interfaces. It may not be known at the moment of accessing these sources exactly what data is available in which source, or the required data may be divided over several different sources. To effectively and efficiently access food knowledge, it is necessary to be able to query across multiple data sources.

A core part of our approach is to store information semantically, so that the meaning of the information is clear. Semantic data is usually stored using RDF [RDF], a W3C standard. There are various types of databases available for storing RDF, within our project we use Sesame [Sesame], free software developed by Aduna which is well suited to applications of the scale which we are developing and which has been widely adopted. Data expressed in RDF can be accessed using SPARQL [SPARQL]. This query language was developed specifically for query semantic data, and is widely used within the semantic community, including the eFoodLab project. It is therefore of essential interest to our activities in the project to know what the state of art and best practices are for 'federated' querying of SPARQL – querying of multiple data sources.

This deliverable discusses what data federation is, with relation to RDF[1] and SPARQL, and describes various ways of achieving federation, with their advantages and disadvantages. It then examines the state of the art in federation technology, and summarises the factors discussed in the literature which can influence federated querying performance.

2 What is federation?

In data federation a query is carried out over multiple data sources. Ideally, the data consumer should not be concerned about these different sources and just view them as a single source. The query must be sent to the different data sources (possibly split into subqueries) and the results combined before they are presented to the user. Technically this is no problem, but it comes at a high cost in terms of performance if done blindly. There are various ways of achieving this in a more efficient manner. Some of these are regarded as 'federation' in the strict sense, others are not called federation but accomplish the same goal. We will discuss these types and what their advantages and disadvantages are.

2.1 Types of federation

2.1.1 'Federation' in application code

Within an application, it is possible to send a query to different data sources. The application itself handles distributing the query and combining the results into one set. This is not a 'federation' in the usual sense of the word, as it is customised to particular queries and particular data sources, but for the end user of the application the effect is the same. If the queries and sources are known beforehand and the number is limited, then this approach can often be attractive in terms of performance, as it can be optimised for the required queries. The sources must also be organised so that they are independent of each other for the desired queries – the results from one should not affect which results are obtained from another.

[1] *Note that there are techniques available for making relational databases available as RDF sources*

2.1.2 *'Federation' in one data source*

If the input sources allow it, it is possible to copy the RDF into one single repository, which can then be queried as normal. This is again not strictly speaking a federation, as the querying is performed on one source. However if the sources are static, or if updates are carried out when data is changed, then the result is, again, the same for the user. This is a simple approach which does not require foreknowledge of the queries or the sources, or additional processing of the results, and which can have high performance. However it is not feasible for large repositories, and some data suppliers do not allow copying the data into a local repository. This approach also assumes that the input sources are stable and well-known.

2.1.3 *SPARQL 1.1 Federation Extensions*

Another form of federation allows the data consumer to include directions to specific input sources in the query itself. The most recent version of the SPARQL language, SPARQL 1.1 [SPARQL], specifically provides support for this type of federated querying. It is possible to specify per section of the query from which source the data should be obtained. Data sources must be SPARQL endpoints. This approach allows querying of multiple sources without any additional processing of the results. The disadvantage is that it is necessary to know exactly which data comes from which source, and to write the queries accordingly. Note that a difference with the application code approach is that the results can influence each other. The extensions can be used on any RDF repository which supports SPARQL 1.1.

In the following example from the SPARQL documentation, persons are obtained from one data source (not visible in the example but assumed to be pre-set for the entire query), and information about their acquaintances from another (http://example.org/sparql).

```
PREFIX foaf:   <http://xmlns.com/foaf/0.1/>

SELECT ?s ?o

{

  ?s a foaf:Person

  SERVICE <http://example.org/sparql> {?s foaf:knows ?o }

}
```

2.1.4 *Federation Engines*

Federation engines are the most sophisticated method of federated querying. These carry out any valid SPARQL query over multiple sources, without requiring any prior knowledge from the data consumer of where the exact data is stored. The query is automatically split into subqueries, which are sent to the different sources, and the results are combined. This has the advantage of allowing a great deal of flexibility in querying. This has however a cost in terms of performance, so a great deal of attention has to be paid to optimisation. The exact approach and how good the performance is, varies per engine. [Rakhmawati] describes how federation engines work. They distinguish a number of basic steps which all federation engines carry out:

- Query parsing
    - The query is split into subqueries
- Choice of sources

- The engine must decide, per subquery, which data sources are relevant. Most engines use one of the following:
  - ASK queries[2]
  - Data catalogues
  - Data indexes
  - Statistical information about the data sources

- Query planning and execution
  - Grouping the queries to reduce the overhead associated with querying each data source
  - Determining the best order to carry out the queries in
  - Joining the results together efficiently

NB: For manual control it is still possible to use the SERVICE keyword from SPARQL 1.1 in a query that is sent to a federation engine. This has the effect of specifying the source for that part of the query instead of letting the federation engine determine it. This can improve the performance.
Some engines are intended for accessing Linked Data rather than for Federation. The difference is that a federated approach selects the sources from the federation members prior to query execution, whereas the Linked Data approach steps from source to source via links in the data, while the query is being executed. A Linked Data approach has the advantage that it can work on any form of linked data, whereas federation usually requires that the sources be SPARQL endpoints. The Linked Data approach is usually less efficient, but work on optimizing this path is in progress [Saleem].

2.2 Available federation engines

A number of federation engines have been built, some of which are available as commercial products, others which are research prototypes. The most promising candidates are described here in more detail; the others are summarised in a list.

2.2.1 Sesame Federation SAIL

Since version 2.7.0, Sesame supports the SPARQL 1.1 extension. In addition to this they have an implementation of their SAIL store as a Federation. There is as yet no official documentation of how this Federation works.
According to Jeen Broekstra, one of the developers, the Sesame Federation SAIL splits the query on each join, and evaluates these in parallel on the federation members. Further, subqueries which can be evaluated on one single federation member are isolated.
There are currently no plans for further development of Sesame Federation. The federation engine was released after the most recent of the overview papers in the literature was published, so no comparison of the performance is available.

2.2.2 FedX

The FedX engine is commercially available from http://www.fluidops.com/fedx/. It is built on top of Sesame, and contains more optimisations. Their article [Schwarte 1] gives an overview of these optimisations, which are discussed further in [Schwarte 2]. A summary of the most important features is:

[2] *An ASK query in SPARQL simply checks if a triple conforming to the pattern in the query is present in the database. ASK queries are usually much faster than queries which return data.*

- Statement sources: SPARQL ASK queries are used to determine the relevant sources
- Parallel processing: Joins and Unions are multithreaded
- Join order: Heuristics are used to determine the cost of each join, and these costs are used to choose the best order
- Bound joins: Joins are calculated in a block to reduce the number of requests
- Groupings: Statements with the same relevant source are grouped

FedX is considered as one of the best solutions, given discussions in the literature. In [Montoya 1] FedX was demonstrated not to cope as well with poor network performance. ANAPSID was better in this respect. Both FedX and ANAPSID were influenced by the data structure, sometimes ANAPSID was better, sometimes FedX. Table 5 in [Montoya 1] showed that the type of query also has a strong effect on which of the two engines performs better. In the tests in [Saleem] FedX came top of the six federations tested.

2.2.3 ANAPSID

The ANAPSID source code is available from github https://github.com/anapsid/anapsid
This engine is described in [ANAPSID]. ANAPSID uses statistics to select the best sources, and a bushy-tree plan to plan the queries. Attention is also paid to handling the situation where a source does not respond (timely). In the measurements in [Montoya 1] ANAPSID is sometimes faster than FedX, sometimes slower.

2.2.4 SPLENDID

The SPLENDID engine is available on the Google Code website http://code.google.com/p/rdffederator/
Sources are selected based on predicates. Statistics about the predicates are available from VOID (Vocabulary Of Interlinked Datasets). If these statistics are insufficient to choose a source, then ASK queries are used. SPLENDID uses dynamic programming with bushy trees to optimise queries. The execution of these queries itself is not yet optimised. According to their own paper, SPLENDID is almost as fast as FedX [SPLENDID].

2.2.5 Semagrow

Semagrow is currently being developed as part of an EU project. They use metadata about the available stores to select the best ones. If necessary then they carry out an ontology alignment step to translate the subquery into the vocabulary of the relevant source. The architecture is described in [Semagrow]. As yet the engine itself is not complete.

2.2.6 SSGS/SSGM

Heuristics and information about the predicates are used to choose the data sources. A bushy-tree plan is used for query execution. Similar times to FedX are measured [Montoya 2] when testing on the FedBench [FedBench] queries. For their own self-designed complex queries their system performs better than FedX, which fails to give results in those cases.

2.2.7 Others

- Aderis – Uses metadata, in the simplest case a list of predicates, to choose sources. Can only handle simple query patterns [Aderis].
- Atlas – Uses an index [Atlas].

- Avalanche – Uses the cardinality of variables (which is dynamically checked), plus the prefixes in the subquery to select the source. The cardinality is used to indicated how selective a subquery is for a given source; the most selective (least distributed) is executed first [Avalanche].
- DARQ – Built on Jena ARQ. Uses indexes of the predicates to choose the best sources. It is no longer being developed but is often used for comparison purposes [DARQ].
- DAW – Uses a combination of an index and ASK queries. Detects where data is duplicated in different datasets [DAW].
- Defender– Built on top of ANAPSID. Uses a predicate list to choose the source, and a bushy-tree plan for query execution. They describe plans for testing and comparing with FedX en DARQ, but these have not yet been carried out. [Defender].
- GDS – Built on Jena. Uses the Kruskal algorithm [GDS].
- Granatum – Uses federation over Linked Data [Granatum].
- Jena ARQ – Uses SPARQL 1.1 and nested inner joins [Jena ARQ].
- LDQPS - Uses federation over Linked data [LDQPS].
- LHD – Sources are chosen from VoID plus using ASK queries. Dynamic programming is used to build the query plan [LHD].
- SemWIQ - Uses a catalogue to choose the sources. The RDF type of each subject must be known [SemWIQ].
- SIHJoin - Uses federation over Linked data. They define a new JOIN operator [SIHJoin].
- SPARQL DQP – Rewrites the queries using rules to optimise them[SPARQL-DQP].
- WoDQA – Built on ARQ. The sources are selected using VOID. They then use federation over Linked data [WoDQA].

[Saleem] gives a good overview of almost all the engines listed here. They also compare the performance of DARQ, SPLENDID, FedX, LHD, ANAPSID and Aderis. FedX gave the best performance.

3 Factors influencing the performance of a federation

In the literature there are a number of factors described which affect the performance of a federated query. The importance of these factors depends for a large part on the approach used by a specific federation engine. Some factors relate to the query, others to how the data sources are structured, and some are relevant for both. The factors have been roughly divided into these three categories; however it is important to remember than they can influence each other – for example, a different data structure requires a different query.

3.1 Factors relating to the query

3.1.1 Number of triple patterns and query shape

Data in RDF is stored in triples – statements consisting of a subject, predicate and object. The effect of the number of the triple patterns is discussed in [Silurian], [Montoya 1]. The more triple patterns, the longer it takes for the query to complete. The form of the query is also important, so-called 'star shapes' are difficult. A 'star shaped query' is a join of multiple BGPs (basic graph patterns) which either share exactly one variable, or which are themselves 'star-shaped'.

3.1.2 Bindings instead of Filters

There are two main methods in SPARQL for searching for statements where a given variable has a specific value. With a Binding, the variable is set to the given value prior to querying. With a Filter, the results are filtered afterwards to match for the value. [Schwarte 2] discusses the difference, and concludes that it is better to use a binding than to filter afterwards.

3.1.3 Instantiations and their position in triple patterns

The position which variables occupy in a triple pattern can make a difference. For example, if the predicate is a variable, then this can increase the query time, as many different endpoints will then match the triple pattern [Montoya 1].

3.1.4 Use of OPTIONAL

The OPTIONAL keyword in SPARQL allows results to be returned even when part of the query finds no result. For example, if a user has a name specified but no email address, then they will still be found by the query if the email address is specified as OPTIONAL. Using OPTIONAL has a large detrimental effect on performance [Montoya 1].

3.2 Factors relating to the data structure

3.2.1 Remote or local

Local repositories are faster as they have no HTTP overhead or network latency. The speed is increased further if the repositories are accessed via their native API [Schwarte 2]. For this same reason repositories which are accessed via their interface perform better than SPARQL endpoints [Rakhmawati].

3.2.2 Availability of concept types

If the concept types are explicitly defined in the data instead of implicitly, then this can help performance [SPLENDID]. For some engines this is even a requirement [SemWIQ].

3.2.3 Using owl:sameAs links

Concepts in different repositories which are essentially the same thing, can be linked using the owl:sameAs relationship. The availability of such links improves the quality of search results. It is better for performance if these relationships are not stored in 3rd party 'link sets' but in the original datasets, as this reduces the number of queries [SPLENDID].

3.2.4 Data partitioning and distribution

How the data is distributed over the repositories, has an effect on performance. In horizontal partitioning the properties are shared over different repositories. In vertical partitioning one data source contains all triples for a single property. Horizontal partitioning affects how complete the result set is. Vertical partitioning affects how long it takes for the query to finish. It also matters if the data is duplicated in different repositories. This can improve results in the situation where a data source drops out, but can increase the time required for the query to complete. [Montoya] [Silurian] [Rakhmawati].

3.2.5 The amount of data

[Schwarte 3] state that a federation can actually be faster than a single central store for large amounts of data. The balance between the advantages of distributed, parallel processing and the disadvantages of the extra communication overhead can then tip in the favour of federation.

3.3 Factors related to both the query and the data structure

3.3.1 Number of relevant sources per subquery

All federation engines select relevant sources for each subquery. The more relevant sources, the longer it takes for the query. The distribution of the data and how the query is written determine how many sources are relevant. Depending on the engine the choice of a source is more or less difficult.

3.3.2 General predicates

General predicates such as rdf:type occur in almost all sources, and make it difficult to choose a source [Silurian]. This is made worse if the source selection is based mainly on predicates.

4 Future work related to federated querying

In the future we are planning to conduct practical experiments on federated querying, to help make informed choices about setting up and using data federations. We will use datasets from real-life examples to help assess the effect of the factors described in the literature. In particular we want to address the question of when a federation is a good choice or not, and what the reasons are to choose for a federation engine or for one of the other three approaches. This aspect has not been examined in the literature. We will also look into how we can translate the information about factors into concrete tips for designing data sources.

5 References

[Aderis] - http://code.google.com/p/sparql-aderis/wiki/Overview

[ANAPSID] - Evaluating Adaptive Query Processing Techniques for Federations of SPARQL Endpoints, Maribel Acosta and Maria-Esther Vidal, 1995
http://iswc2011.semanticweb.org/fileadmin/iswc/Papers/PostersDemos/iswc11pd_submission_95.pdf

[Atlas] – Atlas: Storing, updating and querying rdf(s) data on top of dhts
Z. Kaoudi, M, Koubarakis, I. Miliaraki, M., Magiridou and A. Papadakis-Pesaresi, 2010

[Avalanche] - Avalanche: Putting the Spirit of the Web back into Semantic Web Querying, Cosmin Basca and Abraham Bernstein, 2010
http://ceur-ws.org/Vol-669/ssws2010-paper5.pdf

[DARQ] - Querying Distributed RDF Data Sources with SPARQL, Bastian Quilitz and Ulf Leser, 2008, https://www.sar.informatik.hu-berlin.de/wbi/research/publications/2008/DARQ-FINAL.pdf

[DAW] – Daw: Duplicate-aware federated query processing over the web of data,

M. Saleem, A.C Ngonga Ngomo, J. X. Parreira, H.F. Deus and M. Hauswirth, 2013

[Defender] - DEFENDER: a DEcomposer For quEries agaiNst feDERations of endpoints,
Gabriela Montoya, Maria-Esther Vidal, and Maribel Acosta, 2012
http://2012.eswc-conferences.org/sites/default/files/eswc2012_submission_355.pdf

[GDS] – Querying the web of data with graph theory-based techniques
X. Wang, T. Tiropanis and H. Davis, 2011

[Granatum] - Cataloguing and Linking Life Sciences LOD Cloud,
Ali Hasnain, Ronan Fox, Stefan Decker and Helena F. Deus, 2011
http://www.deri.ie/sites/default/files/publications/cataloguing_and_linking_life_sciences_lod_cloudfinal_resubmission.pdf

[Jena ARQ] - http://jena.apache.org/documentation/query/

[LHD] - LHD: Optimising Linked Data Query Processing Using Parallelisation,
Xin Wang, Xin Wang, Hugh C. Davis, 2013
http://events.linkeddata.org/ldow2013/papers/ldow2013-paper-06.pdf

[LDQPS] - Linked Data Query Processing Strategies,
Günter Ladwig, Thanh Tran, 2010
http://iswc2010.semanticweb.org/pdf/152.pdf

[Montoya 1] - Benchmarking Federated SPARQL Query- Engines: Are Existing Testbeds Enough?
Gabriela Montoya, Maria-Esther Vidal, Oscar Corcho, Edna Ruckhaus, and Carlos Buil-Aranda, 2012,
http://iswc2012.semanticweb.org/sites/default/files/76500306.pdf

[Montoya 2] - A Heuristic-Based Approach for Planning Federated SPARQL Queries,
Gabriela Montoya, Maria-Esther Vidal, and Maribel Acosta, 2012
http://ceur-ws.org/Vol-905/MontoyaEtAl_COLD2012.pdf

[Nikolov] - FedSearch: Efficiently Combining Structured Queries and Full-Text Search in a SPARQL Federation,
Andriy Nikolov, Andreas Schwarte, Christian Hütter, 2013
http://link.springer.com/chapter/10.1007%2F978-3-642-41335-3_27#page-1

[Rakhmawati] - Querying over Federated SPARQL endpoints - A state of the art survey
Nur Aini Rakhmawati, Jürgen Umbrich, Marcel Karnstedt, Ali Hasnain, Michael Hausenblas, 2013
http://arxiv.org/abs/1306.1723 Presentatie met samenvatting van paper

[RDF] - http://www.w3.org/RDF/

[Saleem] – A Fine-Grained Evaluation of SPARQL Endpoint Federation Systems,
Muhammad Saleem, Yasar Khan, Ali Hasnain, Ivan Ermilov, Axel-Cyrille Ngonga Nogomo, 2014
http://www.semantic-web-journal.net/system/files/swj625.pdf

[Schwarte 1] - FedX: A Federation Layer for Distributed Query Processing on Linked Open Data,
Andreas Schwarte, Peter Haase, Katja Hose,Ralf Schenkel, and Michael Schmidt, 2011
http://fluidops.com/download/FedX

[Schwarte 2] - FedX: Optimization Techniques for Federated Query Processing on Linked Data,
Andreas Schwarte, Peter Haase, Katja Hose, Ralf Schenkel, and Michael Schmidt, 2011
http://www.fluidops.com/download/FedX_Optimization+Techniques

[Schwarte 3] – An Experiende Report of Large Scale Federations,
Andreas Schwarte, Peter Haase, Michael Schmidt, Katja Hose, and Ralf Schenkel, 2012
http://arxiv.org/pdf/1210.5403v1.pdf

[Semagrow]- D3.4.1: Techniques for Heterogeneous Distributed Semantic Querying, 2014
http://semagrow.eu/sites/default/files/D3.4.1-Techniques%20for%20Heterogeneous%20Distributed%20Semantic%20Querying.pdf

[SemWIQ] SemWIQ - Semantic Web Integrator and Query Engine,
Andreas Langegger and Wolfram Wöß, 2014
https://www.researchgate.net/publication/221387725_SemWIQ_-_Semantic_Web_Integrator_and_Query_Engine

[Sesame] - http://www.openrdf.org/

[Sesame Federation] -
https://bitbucket.org/openrdf/sesame/src/f245952347b0b11be8e8740bdfd8beddb636b105/core/sail/federation/src/main/java/org/openrdf/sail/federation/evaluation/?at=2.7.x

[SIHJoin] - SIHJoin: Querying Remote and Local Linked Data – Technical Report,
Günter Ladwig, Thanh Tran, 2011
http://people.aifb.kit.edu/gla/tr/sq_report.pdf

[Silurian] - Silurian - a sparql visualiser for understanding queries and federation,
Simón Castillo, Guillermo Palma, Maria-Esther Vidal, 2013
http://iswc2013.semanticweb.org/sites/default/files/iswc_demo_35.pdf

[SPARQL] - http://www.w3.org/TR/sparql11-query/

[SPARQL-DQP]- Semantics and optimisation of the SPARQL 1.1 federation extension,
Carlos Buil-Aranda, Marcelo Arenas and Oscar Corcho, 2011
http://mayor2.dia.fi.upm.es/oeg-upm/files/pdf/eswc11paper.pdf

[SPLENDID] - SPLENDID: SPARQL Endpoint Federation Exploiting VOID Descriptions
Olaf Gorlitz and Steffen Staab, 2011
http://userpages.uni-koblenz.de/~staab/Research/Publications/2011/COLD2011-SPLENDID-1.pdf

[WoDQA] - Querying the Web of Interlinked Datasets
using VOID Descriptions,
Ziya Akar, Tayfun Gökmen Halaç, Erdem Eser Ekinci, Oguz Dikenelli, 2012,
http://events.linkeddata.org/ldow2012/papers/ldow2012-paper-06.pdf

